INF1010 - Programmation orientée objet

Commencé le	vendredi 1 mars 2019, 18:30
État	Terminé
Terminé le	vendredi 1 mars 2019, 19:47
Temps mis	1 heure 16 min
Points	38,47/44,00
Note	**17,49** sur 20,00 (**87**%)

NAVIGATION DU TEST

Question en vrac
1 2 3 4 5 6

Héritage et affichage
7

Vecteur et surcharge des opérateurs
8

Héritage et conversion
i 9 10 11 12 13

Composition et Agrégation
i 14 15 16 17

Afficher une page à la fois

Terminer la relecture

Question **1**
Partiellement correct
Note de 1,50 sur 2,00
Marquer la question

Quelle(s) est (sont) les instructions valides selon une méthode constante ?

Attention: on retire des points par mauvaise réponse

Veuillez choisir au moins une réponse :

☐ a.
```
void A::méthode() const { this->attribut_ = 10; }
```

☑ b.
```
void A::méthode() const {

    int x = 1;

    x = 4;

}
```
✓

☑ c.
```
void B::méthodeDeB() { }

void A::méthode() const {

    B b;

    b.méthodeDeB();

}
```
✓

☑ d.
```
void A::méthode() const { }

void fonction(const A& a) { a.méthode(); }
```
✓

☐ e.
```
void B::méthodeDeB() { }

void A::méthode() const { this->b_.méthodeDeB(); }
```

☐ f.
```
void fonction (const A& a) { }

void A::méthode() const { fonction(*this); }
```

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 3.
Les réponses correctes sont :
```
void fonction (const A& a) { }

void A::méthode() const { fonction(*this); }

,
void A::méthode() const { }

void fonction(const A& a) { a.méthode(); }

,
void B::méthodeDeB() { }

void A::méthode() const {

    B b;

    b.méthodeDeB();

}

,
void A::méthode() const {

    int x = 1;

    x = 4;

}
```

Question **2**
Correct
Note de 1,00 sur 1,00
Marquer la question

En lisant la définition d'une classe, vous trouvez l'attribut **b_** suivant:

```
class A
{
//...
private:
    B* b_;
};
```

Quel(s) énoncé(s) sont possibles par rapport à l'allocation du pointeur **b_**?

Veuillez choisir au moins une réponse :

- ☐ a. Il correspond à un tableau de pointeurs réservé dans le tas où chaque élément pointe vers un espace mémoire de type **B** dans le tas.
- ☐ b. Il correspond à un tableau de pointeurs réservé dans la pile, mais qui pointe vers un objet de type **B** dans le tas.
- ☑ c. Il pointe vers un seul objet de type **B** réservé dans le tas. ✓
- ☐ d. Il correspond à un tableau de pointeurs vers des objets de type **B** réservé dans la pile.
- ☑ e. Il pointe vers un tableau d'objets de type B réservé dans le tas. ✓
- ☑ f. Il pointe vers un objet de type **B** réservé dans la pile. ✓

Votre réponse est correcte.

Les réponses correctes sont : Il pointe vers un seul objet de type **B** réservé dans le tas., Il pointe vers un objet de type **B** réservé dans la pile., Il pointe vers un tableau d'objets de type B réservé dans le tas.

Question **3**
Correct
Note de 1,00 sur 1,00
Marquer la question

Pour respecter le principe d'encapsulation, il faut toujours définir un accesseur (getter) et un mutateur (setter) pour chaque attribut d'une classe.

Sélectionnez une réponse :

○ Vrai

● Faux ✓

Faux, car le principe d'encapsulation nous permet de DECIDER si un utilisateur de la classe à le droit d'accéder ou de modifier un attribut.

Si le principe d'encapsulation nous oblige de définir un accesseur et un mutateur pour chaque attribut d'une classe, alors aussi bien mettre tous les attributs publics.

La réponse correcte est « Faux ».

Question **4**
Incorrect
Note de 0,00 sur 2,00
Marquer la question

Soit le code suivant:

```
#include <iostream>
using namespace std;

class MatelasBase {
public:
    MatelasBase() {}

    double getPrix() const {  return 1000;  }
};

class MatelasSophistique : public MatelasBase {
public:
    MatelasSophistique(double facteur): facteur_(facteur) {}

    double getPrix() const {
       // À implémenter
    }
private:
    double facteur_;
};

int main() {
    MatelasSophistique ms(1.2);
    cout << ms.getPrix() << "$"; // Devrait afficher 1200$;
}
```

Écrivez l'implémentation de la méthode getPrix() de **MatelasSophistique** (l'unique ligne de code à placer comme corps de la méthode) qui applique le facteur sur le prix indiqué par la classe de base.

Réponse : return facteur_ * getPrix(); ✗

La réponse correcte est : return MatelasBase :: getPrix() * facteur_

Question **5**
Correct
Note de 1,00 sur 1,00
Marquer la question

Soit l'initialisation de la variable **a** suivante.

```
A* a[10];
for (size_t i = 0; i < 10; i++)
    a[i] = new A();
```

Quelle instruction désalloue correctement la mémoire?

Veuillez choisir une réponse :

● a.
```
for (int i = 0; i < 10; i++)
    delete a[i];
```
✓

○ b.
```
for (int i = 0; i < 10; i++)
    delete[] a[i];
```

○ c.
```
for (int i = 0; i < 10; i++)
    delete[] a[i];
delete[] a;
```

○ d.
```
delete a;
for (int i = 0; i < 10; i++)
    delete[] a[i];
```

○ e.
```
delete a;
```

○ f.
```
for (int i = 0; i < 10; i++)
    delete[] a[i];
delete a;
```

○ g.
```
delete[] a;
```

○ h.
```
delete[] a;
for (int i = 0; i < 10; i++)
    delete[] a[i];
```

Votre réponse est correcte.

La réponse correcte est :
```
for (int i = 0; i < 10; i++)
    delete a[i];
```

Question **6**
Correct
Note de 1,00 sur 1,00
Marquer la question

Combien de fois le constructeur par copie est appelé dans le code suivant?

```
#include <iostream>
using namespace std;

class MaClasse {
public:
    MaClasse() {}
    MaClasse(const MaClasse& c) {  cout << "Copie!" << endl;  }
};

void f1(MaClasse& c) {
   MaClasse c2;
}

MaClasse& f2(MaClasse i1, MaClasse i2) {
    MaClasse* i = new MaClasse();
    return *i;
}

int main() {
    MaClasse c1 = MaClasse();
    MaClasse c2 = c1;
    MaClasse c3;
    c3 = c2;
    MaClasse& c4 = f2(c1, c2);
    f1(c4);
}
```

Réponse : 3 ✓

La réponse correcte est : 3

Question **7**
Correct
Note de 2,00 sur 2,00
Marquer la question

Soit le code C++ suivant :

```
#include <iostream>

using namespace std;

class A {
public:
    A() { cout << "A() "; }
private:
    int attC_;
};

class C {
public:
    C() { cout << "C() "; }
private:
    int attE_;
};

class B {
public:
    B() {
        cout << "B() ";
        pA_ = new A();
    }
    ~B() {
        delete pA_;
    }
private:
    C  attB_;
    A* pA_;
};

class D : public B {
public:
    D() { cout << "D() "; }

};

void main() {
    // 1
    B* p1 = new B();
    cout << endl;
    // 2
    D* p2 = new D();
    cout << endl;
}
```

1- Quel est l'affichage de l'instruction 1 (B* p1 = new B();) ?

C() ✓ B() ✓ A() ✓ [Rien] ✓

2- Quel est l'affichage de l'instruction 2 (D* p2 = new D();) ?

C() ✓ B() ✓ A() ✓ D() ✓

3- Identifiez les différents types de relations entre les classes A , B , C et D

Entre A et B : Composition ✓
Entre A et C : Pas de relation ✓
Entre B et C : Composition ✓
Entre B et D : Héritage ✓

Votre réponse est correcte.

La réponse correcte est :
Soit le code C++ suivant :

```
#include <iostream>

using namespace std;

class A {
public:
    A() { cout << "A() "; }
private:
    int attC_;
};

class C {
public:
    C() { cout << "C() "; }
private:
    int attE_;
};

class B {
public:
    B() {
        cout << "B() ";
        pA_ = new A();
    }
    ~B() {
        delete pA_;
    }
private:
    C  attB_;
    A* pA_;
};

class D : public B {
public:
    D() { cout << "D() "; }

};

void main() {
    // 1
    B* p1 = new B();
    cout << endl;
    // 2
    D* p2 = new D();
    cout << endl;
}
```

1- Quel est l'affichage de l'instruction 1 (B* p1 = new B();) ?

`[C()][B()][A()][[Rien]]`

2- Quel est l'affichage de l'instruction 2 (D* p2 = new D();) ?

`[C()][B()][A()][D()]`

3- Identifiez les différents types de relations entre les classes A , B , C et D

```
Entre A et B : [Composition ]
Entre A et C : [Pas de relation]
Entre B et C : [Composition ]
Entre B et D : [Héritage]
```

Question **8**
Partiellement correct
Note de 8,47 sur 10,00
Marquer la question

Votre ami Patrice Zenforme écrit un programme pour calculer le nombre de calories de ses repas.

Il vous montre les classes suivantes qu'il a commencé à écrire:

```
#include <iostream>
#include <vector>
using namespace std;

class Ingredient {
public:
  Ingredient(string nom, int nbCalories);
  int getNbCalories() const;

private:
  string nom_;
  int nbCalories_;
};

Ingredient::Ingredient(string nom, int nbCalories)
  : nom_(nom), nbCalories_(nbCalories) {}

int Ingredient::getNbCalories() const {
  return nbCalories_;
}


class Repas {
public:

private:
  vector<Ingredient*> ingredients_;
};

int main() {
  Ingredient* i1 = new Ingredient("Banane", 100);
  Ingredient* i2 = new Ingredient("Fromage chedar", 200);
  Ingredient* i3 = new Ingredient("Sauce brune", 100);
  Ingredient* i4 = new Ingredient("Patate", 300);

  // On ajoute les ingrédients d'une poutine
  Repas poutine;
  (((poutine += i1) += i2) += i3) += i4;

  // Oups, on a fait une erreur. On a rajouté une banane...
  poutine -= i1;
  cout << poutine << endl;

  delete i1;
  delete i2;
  delete i3;
  delete i4;
}
```

Il aimerait simplifier l'utilisation de ces classes en y intégrant la surcharge d'opérateurs de C++.

1. Surcharge de l'opérateur += de la classe Repas

Implémentez la surcharge de l'opérateur += de la classe **Repas** qui ajoute un ingrédient dans le vecteur faisant une agrégation.

- Déclaration: (Dans la partie "public" de la classe Repas ✓)

```
[rien] ✓ Repas& ✓ [rien] ✓ operator+=( Ingredient* ingredient ✓ ) [rien] ✓ ;
```

- Implémentation:

```
[rien] ✓ Repas& ✓ Repas:: ✓ operator+=( Ingredient* ingredient ✓ ) [rien] ✓ {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    ingredients_.push_back(ingredient); ✓
    return *this; ✓
}
```

2. Surcharge de l'opérateur -= de la classe Repas

Implémentez la surcharge de l'opérateur -= de la classe **Repas** qui retire un ingrédient du vecteur **ingredients_**.

Lorsque vous supprimez un élément du vecteur, NE tenez PAS en compte l'ordre des éléments.

- Déclaration: (Dans la partie "public" de la classe Repas ✓)

```
[rien] ✓ Repas& ✓ [rien] ✓ operator-=( Ingredient* ingredient ✓ ) [rien] ✓ ;
```

- Implémentation:

```
[rien] ✓ Repas& ✓ Repas:: ✓ operator-=( Ingredient* ingredient ✓ ) [rien] ✓ {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    for (size_t i = 0; i < ingredients_.size() ✓ ; i++) {

        if ( ingredients_[i] == ingredient ✓ ) {
             ingredients_.erase(ingredient); ✗
             break; ✗
             [rien] ✗
        }
    }

    return *this; ✓
}
```

3. Surcharge de l'opérateur << de la classe Ingredient

Implémentez la surcharge de l'opérateur << qui permet d'afficher un **Ingredient** vers un flux de sortie.

Par exemple, l'instruction "*cout << *i1*" serait:

```
- Banane => 300 calories
```

- Déclaration: (Dans la partie "public" de la classe Ingredient ✓)

```
friend ✓ ostream& ✓ [rien] ✓ operator<<( ostream& os, const Ingredient& ingredient ✓ ) const ✗ ;
```

- Implémentation:

```
[rien] ✓ ostream& ✓ Ingredient:: ✗ operator<<( ostream& os, const Ingredient& ingredient ✓ ) const ✗ {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    os << ingredient.nom_ ✓
       << " => "
       << ingredient.nbCalories_ ✓
       << " calories";

    return os; ✓
}
```

4. Surcharge de l'opérateur << de la classe Repas

Implémentez la surcharge de l'opérateur << qui permet d'afficher un **Repas** vers un flux de sortie en utilisant l'opérateur << de **Ingredient**.

Par exemple, l'affichage de "*cout << poutine*" serait:

```
Ingrédients:
  - Patate => 300 calories
  - Fromage cheddar => 200 calories
  - Sauce brune => 100 calories

Total: 600 calories
```

- Déclaration: (Dans la partie "public" de la classe Repas ✓)

```
friend ✓ ostream& ✓ [rien] ✓ operator<<( ostream& os, const Repas& repas ✓ ) const ✗ ;
```

- Implémentation:

```
[rien] ✓ ostream& ✓ Repas:: ✗ operator<<( ostream& os, const Repas& repas ✓ ) const ✗ {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    os << "Ingrédients:" << endl;

    int totalCalories = 0;
    for (size_t i = 0; i < repas.ingredients_.size() ✓ ; i++) {
        totalCalories += repas.ingredients_[i]->getNbCalories(); ✓
        os << " - " << *repas.ingredients_[i] ✓ << endl;
    }

    os << "Total: " << totalCalories << " calories";

    return os; ✓
}
```

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 50.
La réponse correcte est :
Votre ami Patrice Zenforme écrit un programme pour calculer le nombre de calories de ses repas.

Il vous montre les classes suivantes qu'il a commencé à écrire:

```
#include <iostream>
#include <vector>
using namespace std;

class Ingredient {
public:
  Ingredient(string nom, int nbCalories);
  int getNbCalories() const;

private:
  string nom_;
  int nbCalories_;
};

Ingredient::Ingredient(string nom, int nbCalories)
  : nom_(nom), nbCalories_(nbCalories) {}

int Ingredient::getNbCalories() const {
  return nbCalories_;
}


class Repas {
public:

private:
  vector<Ingredient*> ingredients_;
};

int main() {
  Ingredient* i1 = new Ingredient("Banane", 100);
  Ingredient* i2 = new Ingredient("Fromage chedar", 200);
  Ingredient* i3 = new Ingredient("Sauce brune", 100);
  Ingredient* i4 = new Ingredient("Patate", 300);

  // On ajoute les ingrédients d'une poutine
  Repas poutine;
  (((poutine += i1) += i2) += i3) += i4;

  // Oups, on a fait une erreur. On a rajouté une banane...
  poutine -= i1;
  cout << poutine << endl;

  delete i1;
  delete i2;
  delete i3;
  delete i4;
}
```

Il aimerait simplifier l'utilisation de ces classes en y intégrant la surcharge d'opérateurs de C++.

1. Surcharge de l'opérateur += de la classe Repas

Implémentez la surcharge de l'opérateur += de la classe **Repas** qui ajoute un ingrédient dans le vecteur faisant une agrégation.

- Déclaration: ([Dans la partie "public" de la classe Repas])

```
[[rien]] [Repas&] [[rien]]operator+=([Ingredient* ingredient]) [[rien]];
```

- Implémentation:

```
[[rien]] [Repas&] [Repas::]operator+=([Ingredient* ingredient]) [[rien]] {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    [ingredients_.push_back(ingredient);]
    [return *this;]
}
```

2. Surcharge de l'opérateur -= de la classe Repas

Implémentez la surcharge de l'opérateur -= de la classe **Repas** qui retire un ingrédient du vecteur **ingredients_**.
Lorsque vous supprimez un élément du vecteur, NE tenez PAS en compte l'ordre des éléments.

- Déclaration: ([Dans la partie "public" de la classe Repas])

```
[[rien]] [Repas&] [[rien]]operator-=([Ingredient* ingredient]) [[rien]];
```

- Implémentation:

```
[[rien]] [Repas&] [Repas::]operator-=([Ingredient* ingredient]) [[rien]] {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    for (size_t i = 0; i < [ingredients_.size()]; i++) {
        if ([ingredients_[i] == ingredient]) {
            [ingredients_[i] = ingredients_.back();]
            [ingredients_.pop_back();]
            [break;]
        }
    }

    [return *this;]
}
```

3. Surcharge de l'opérateur << de la classe Ingredient

Implémentez la surcharge de l'opérateur << qui permet d'afficher un **Ingredient** vers un flux de sortie.

Par exemple, l'instruction "*cout << *i1*" serait:

```
- Banane => 300 calories
```

- Déclaration: ([Dans la partie "public" de la classe Ingredient])

```
[friend] [ostream&] [[rien]]operator<<([ostream& os, const Ingredient& ingredient]) [[rien]];
```

- Implémentation:

```
[[rien]] [ostream&] [[rien]]operator<<([ostream& os, const Ingredient& ingredient]) [[rien]] {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    os << [ingredient.nom_]
       << " => "
       << [ingredient.nbCalories_]
       << " calories";

    [return os;]
}
```

4. Surcharge de l'opérateur << de la classe Repas

Implémentez la surcharge de l'opérateur << qui permet d'afficher un **Repas** vers un flux de sortie en utilisant l'opérateur << de **Ingredient**.

Par exemple, l'affichage de "*cout << poutine*" serait:

```
Ingrédients:
  - Patate => 300 calories
  - Fromage cheddar => 200 calories
  - Sauce brune => 100 calories

Total: 600 calories
```

- Déclaration: ([Dans la partie "public" de la classe Repas])

```
[friend] [ostream&] [[rien]]operator<<([ostream& os, const Repas& repas]) [[rien]];
```

- Implémentation:

```
[[rien]] [ostream&] [[rien]]operator<<([ostream& os, const Repas& repas]) [[rien]] {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    os << "Ingrédients:" << endl;

    int totalCalories = 0;
    for (size_t i = 0; i < [repas.ingredients_.size()]; i++) {
        totalCalories += [repas.ingredients_[i]->getNbCalories();]
        os << " - " << [*repas.ingredients_[i]] << endl;
    }

    os << "Total: " << totalCalories << " calories";

    [return os;]
}
```

Description
Marquer la question

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class ArretAutobus {
public:
    ArretAutobus() {}
    ArretAutobus(string rue) : rue_(rue) {}
private:
    string rue_;
};

class LigneAutobus {
public:
    LigneAutobus(int numero) : numero_(numero) {}
    bool ajouterArret(ArretAutobus* a) {
      // À implémenter
    }
private:
    vector<ArretAutobus*> arrets_; // agrégation de pointeurs à des objets ArretAutobus
    int numero_;
};

class Vehicule {
public:
    Vehicule(int anneeFabrication = 2019) {}
    int getAnneeFabrication() {
      return anneeFabrication_;
    }
private:
    int anneeFabrication_;
};

class Autobus : public Vehicule {
public:
    Autobus() : ligne_(nullptr) {}
    Autobus(char id, int annee = 2019); // À implémenter

    void setLigne(LigneAutobus* ligne) { ligne_ = ligne; }
    char getId() { return id_; }
private:
    LigneAutobus* ligne_; // agrégation d'une ligne d'autobus
    char id_;
};

class ReseauAutobus  {
public:
    ReseauAutobus(int nbAutobus = 1) {
      for (char i = 'A'; i < 'A' + nbAutobus; i++)
        autobus_.push_back(new Autobus(i));
    }
    ~ReseauAutobus() {
      // À implémenter
    }
    void ajouterAutobus(const Autobus& autobus) {
      autobus_.push_back(new Autobus(autobus));
    }

    bool setLigneAutobus(char id, LigneAutobus* ligne) {
      for (size_t i = 0; i < autobus_.size(); i++)
        if (autobus_[i]->getId() == id) {
          autobus_[i]->setLigne(ligne);
          return true;
        }
      return false;
    }
private:
    vector<Autobus*> autobus_; // composition d'autobus
};
```

Question **9**
Terminer
Note de 3,00 sur 4,00
Marquer la question

Soient les classes ci-dessus et la fonction **main**.

Écrivez les déclarations et les instructions suivantes:

```
int main() {
    // déclarez et réservez dynamiquement 4 arrêts d'autobus: arret1, arret2, arret3, arret4
    // Dec/Ed_Montpetit, Ed_Montpetit/VdIndy, CStCatherine/LCollin, Ed_MonPetit/UDM

    // déclarez et réservez automatiquement 2 lignes d'autobus: 51 et 129

    // ajoutez l'arret1 et l'arret3 à la ligne 51

    // ajoutez l'arret1, l'arret2 et l'arret4 à la ligne 129;

    // Déclarez un réseau d'autobus nommé 'stm' avec 15 autobus.

    // Associer la ligne 51 à l'autobus 'D' sur le reseau STM

    return 0;
}
```

```
int main(){

    // déclarez et réservez dynamiquement 4 arrêts autobus:



    ArretAutobus* arret1("Dec/Ed_Montpetit");

    ArretAutobus* arret2("Ed_Montpetit/VdIndy");

   ArretAutobus* arret3("CStCatherine/LCollin");

   ArretAutobus* arret4("Ed_MonPetit/UDM");

    LigneAutobus ligne129(129);

     LigneAutobus ligne51(51);

    ligne51.ajouterArret(arret1);

    ligne51.ajouterArret(arret3);

    ligne129.ajouterArret(arret2);

    ligne129.ajouterArret(arret4);

    ReseauAutobus stm(15);

    stm.setLigneAutobus('D', ligne51);

    return 0;

}
```

Commentaire :
Oublier 'new' pour les arrets -0.5

setLigneAutobus prend un pointeur comme 2ieme paramètre -0.5

Question **10**
Terminer
Note de 5,50 sur 6,00
Marquer la question

Dans la classe **LigneAutobus**, écrire l'implémentation de la méthode suivante:

`bool LigneAutobus::ajouterArret(ArretAutobus* a)`

1. Cette méthode ajoute un arrêt dans l'attribut **arrets_**.

- Si l'arrêt n'existe pas encore sur la ligne, on l'ajoute et on retourne la valeur vraie.
- Si l'arrêt existe, alors on retourne la valeur false.
- Un arrêt est considéré existant si un arrêt déjà présent a le même nom de rue.

2 A-t-on besoin de surcharger un opérateur pour la classe **ArretAutobus**?
Si oui, écrire l'implémentation de cette méthode.

```
// Implémentation de ajouterArret
bool LigneAutobus::ajouterArret(ArretAutobus* a)
{
    for (size_t i = 0; i < arrets_.size(); i++)
    {
        if (*(arrets_[i]) == *(a))
            return false;
    }

    arrets_.push_back(a);
}

// S'il y a lieu, mettre la définition et l'implémentation de la surcharge d'opérateur ici
// Déclaration dans la partie publique de ArretAutobus.h
bool operator==(const ArretAutobus& arretAutobus) const;

// Definition de la fonction membre
bool ArretAutobus::operator==(const ArretAutobus& arretAutobus) const
{
    return arretAutobus.rue_== rue_;
}
```

Commentaire :
return true manquant -0.5

Question **11**
Terminer
Note de 3,00 sur 3,00
Marquer la question

Implémentez le constructeur par paramètres de la classe **Autobus**.

```
Autobus::Autobus(char id, int annee) : Vehicule(annee), ligne_(nullptr), id_(id)
{
}
```

Commentaire :

Question **12**
Terminer
Note de 2,00 sur 2,00
Marquer la question

Écrivez l'implémentation du destructeur de **ReseauAutobus**.

Les classes **ReseauAutobus** et **Autobus** sont reliés par la composition.

```
ReseauAutobus::~ReseauAutobus()
{
    for (size_t i = 0; i < autobus_.size(); i++)
    {
        delete autobus[i];
    }
}
```

Commentaire :

Question **13**
Correct
Note de 1,00 sur 1,00
Marquer la question

Soit le code suivant:

```
vector<Vehicule*> vehicules;
vehicules.push_back(new Autobus('A'));

// Ligne à mettre ici
```

Écrivez la ligne de code qui affiche l'id de l'unique autobus du vecteur vehicules en appelant la méthode **getId()**.

Réponse : cout << static_cast<Autobus*>(vehicules[0])->getId() << endl; ✓

La réponse correcte est : cout << static_cast <Autobus*>(vehicules[0]) ->getId()

Description
Marquer la question

Soient les classes

```
class Cours {
public:
    Cours();
private:
    //....
};

class Semaine {
public:

private:
};
```

Question **14**
Correct
Note de 2,00 sur 2,00
Marquer la question

L'attribut de la classe **Semaine** est **Cours* cours_** et la classe **Semaine** est composite de la classe **Cours**.

Identifier les définitions et implémentations possibles de constructeur de la classe **Semaine**.

Si vous identifiez une fausse réponse, des points vous seront retirés.

Veuillez choisir au moins une réponse :

☐ a. `Semaine(Cours cours) { cours_ = cours; }`

☑ b. `Semaine() { cours_ = new Cours(); }` ✓

☑ c. `Semaine(const Cours& cours) { cours_ = new Cours(cours); }` ✓

☐ d. `Semaine(Cours cours) : cours_(&cours) {}`

☐ e. `Semaine(const Cours& cours) { cours_ = cours; }`

☑ f. `Semaine(Cours* cours) { cours_ = cours; }` ✗

☑ g. `Semaine(Cours* cours) { cours_ = new Cours(*cours); }` ✓

Votre réponse est correcte.

Les réponses correctes sont :
```
Semaine(Cours* cours) { cours_ = new Cours(*cours); }

,
Semaine(const Cours& cours) { cours_ = new Cours(cours); }

,
Semaine() { cours_ = new Cours(); }
```

Question **15**
Correct
Note de 2,00 sur 2,00
Marquer la question

L'attribut de la classe **Semaine** est **Cours cours_** et la classe **Semaine** est composite de la classe **Cours**.

Identifier les définitions et 'implémentations possibles du constructeur de la classe **Semaine**.

Si vous identifiez une fausse réponse, des points vous seront retirés.

Veuillez choisir au moins une réponse :

☑ a. `Semaine(const Cours& cours) { cours_ = cours; }` ✓

☑ b. `Semaine(Cours* cours) { cours_ = *cours; }` ✓

☑ c. `Semaine(const Cours& cours) : cours_(cours) {}` ✓

☑ d. `Semaine() {}` ✓

☐ e. `Semaine(const Cours& cours) { cours_ = new Cours(cours); }`

☐ f. `Semaine(Cours* cours) { *cours_ = *cours; }`

☐ g. `Semaine(Cours* cours) { cours_ = new Cours(*cours); }`

Votre réponse est correcte.

Les réponses correctes sont :
```
Semaine(const Cours& cours) : cours_(cours) {}

,
Semaine(const Cours& cours) { cours_ = cours; }

,
Semaine(Cours* cours) { cours_ = *cours; }

,
Semaine() {}
```

Question **16**
Correct
Note de 2,00 sur 2,00
Marquer la question

L'attribut de la classe **Semaine** est **Cours& cours_** et la classe **Semaine** est une agrégation de la classe **Cours**.

Identifier les définitions et implémentations possibles du constructeur de la classe **Semaine**.

Si vous identifiez une fausse réponse, des points vous seront retirés.

Veuillez choisir au moins une réponse :

☐ a. `Semaine(Cours& cours) : cours_(&cours) {}`

☑ b. `Semaine(Cours& cours) : cours_(cours) {}` ✓

☐ c. `Semaine() : cours_(nullptr) {}`

☐ d. `Semaine(Cours* cours) : cours_(new Cours(*cours)) {}`

☐ e. `Semaine(Cours cours) : cours_(cours) {}`

☐ f. `Semaine(Cours& cours) { cours_ = cours; }`

Votre réponse est correcte.

La réponse correcte est :
`Semaine(Cours& cours) : cours_(cours) {}`

Question **17**
Correct
Note de 2,00 sur 2,00
Marquer la question

L'attribut de la classe **Semaine** est **Cours* cours_** et la classe **Semaine** est une agrégation de la classe **Cours**.

Identifier les définitions et implémentations possibles du constructeur de la classe **Semaine**.

Veuillez choisir au moins une réponse :

☐ a. `Semaine(Cours cours) { cours_= &cours; }`

☐ b. `Semaine(Cours* cours) { *cours_ = *cours; }`

☐ c. `Semaine(Cours* cours) { cours_= &cours; }`

☑ d. `Semaine(Cours* cours) : cours_(cours) {}` ✓

☑ e. `Semaine(Cours* cours) { cours_= cours; }` ✓

☐ f. `Semaine(Cours* cours) { cours_= new Cours(); *cours_ = *cours; }`

☐ g. `Semaine(Cours* cours) { cours_ = new Cours(*cours); }`

Votre réponse est correcte.

Les réponses correctes sont :
```
Semaine(Cours* cours) { cours_= cours; }

,
Semaine(Cours* cours) : cours_(cours) {}
```

Terminer la relecture

◄ Annonces

Aller à... ▾